



**DEPARTAMENTO
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA

Primer Trabajo Práctico

14 de Abril de 2010

Algoritmos y Estructuras de Datos III

| Integrante | LU | Correo electrónico |
|--------------------------------|--------|-----------------------------|
| Bianchi, Mariano | 92/08 | bianchi-mariano@hotmail.com |
| Brusco, Pablo | 527/08 | pablo.brusco@gmail.com |
| Di Pietro, Carlos Augusto Lyon | 126/08 | cdipietro@dc.uba.ar |



Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.fcen.uba.ar>

Índice

Ejercicio 1

Introducción

El primer problema del presente trabajo consistió en la implementación de un algoritmo capaz de dar solución a la ecuación

$$b^n \bmod (n) \quad (1)$$

haciendo uso de alguna de las técnicas algorítmicas aprendidas hasta el momento en la materia. Asimismo, la consigna dictaba que la complejidad final del algoritmo debería ser menor a $O(n)$. En pos de cumplimentar lo pedido se decidió usar la técnica de *Dividir & Conquistar*¹ para desarrollar el algoritmo. Esta técnica se caracteriza principalmente en dividir la instancia de un problema en instancias más pequeñas, atacar cada una de ellas por separado y resolverlas, para finalmente juntar sus resultados y así producir el resultado final.

Detalles de implementación

La primera solución que se piensa casi de manera intuitiva es la de mutliplicar n veces el número b y luego hallar el resto de dividir ese resultado por n .

$$\underbrace{b.b.b \dots b.b.b}_{n \text{ veces}} \bmod (n) = b^n \bmod (n) \quad (2)$$

Sin embargo, la complejidad ese algoritmo es $O(n)$, ya que se realizan n multiplicaciones y 1 división, razón por lo cual no cumple con lo pedido en la consigna.

No obstante, la idea anterior conduce a otra forma de encarar el problema. Veamos de qué se trata.

Sabemos que $b^n \bmod (n)$ es el resto de dividir b^n por n . Llamemos x a ese resto. Luego:

$$x = b^n \bmod (n) , \quad \text{con } 0 \leq x < n \quad (3)$$

ya que el Algoritmo de División de Números Enteros asegura que el resto existe, es único y es un valor entre 0 y n .

Inmediatamente de lo anterior, por la definición de congruencia, se desprende que:

$$b^n \equiv x \pmod{n} \quad (4)$$

Luego, una de las propiedades de congruencias, nos asegura que:

$$a \equiv p(n) , \quad b \equiv q(n) \implies a.b \equiv p.q(n) , \quad \forall a, b, p, q, n \in \mathbf{Z} \quad (5)$$

por lo que, tomando en particular, $a = b$ sigue que:

$$b \equiv p(n) \implies b.b \equiv p.p \Leftrightarrow b^2 \equiv p^2(n) \quad (6)$$

Sea ahora k el resto de la division de p^2 por n ; luego:

$$p^2 \equiv k \pmod{n} , \quad \text{con } 0 \leq k < n \quad (7)$$

¹Poner alguna referencia en donde se explique esta técnica

Por lo tanto, de 6 y 7 se obtiene usando la propiedad transitiva de congruencias que:

$$b^2 \equiv p^2(n) , \ p^2 \equiv k(n) \implies b^2 \equiv k(n) \quad (8)$$

donde por 7 sabíamos que k es un número entre 0 y n .

De este modo, podemos aplicar el resultado de 8 en 4, obteniéndose:

$$b^n = \underbrace{b.b.b \dots b.b.b}_{n \text{ veces}} \equiv x(n)b^n = \underbrace{b^2.b^2 \dots b^2.b^2}_{n/2 \text{ veces}} \equiv x(n)b^n = \underbrace{k.k \dots k.k}_{n/2 \text{ veces}} \equiv x(n) \quad (9)$$

Ejercicio 2

Introducción

El segundo problema consistió en la implementación de un algoritmo capaz de dar solución al problema que se plantea a continuación:

Decidir si un grupo de n personas pueden formar o no una ronda que cumpla con las siguientes restricciones

- La ronda debe contener a todas las personas.
- Algunas personas son amigas y otras no.
- Cada alumna debe tomar de la mano a dos de sus amigas.

Detalles de implementación

Dado que para este problema no se conocen algoritmos buenos², se pensó en utilizar la solución por fuerza bruta, con algunas mejoras, es decir, intentar todas las combinaciones hasta lograr determinar si hay o no solución.

²Un algoritmo se considera bueno si puede ser resuelto en tiempo polinomial.